



Analysis of path exclusions at the assembly level

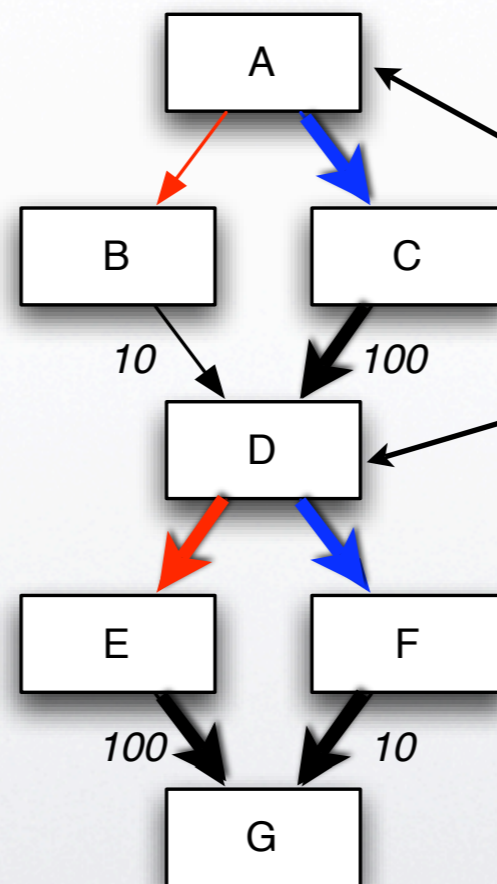
7th Int'l Workshop on
Worst-Case Execution Time (WCET) Analysis

Ingmar Stein, Florian Martin
ingmar@absint.com



The Goal

200



Jump conditions
are equivalent

110



The Goal (2)

- ```
if (flag)
 do_something_expensive();

/* ... */

if (flag)
 do_something_cheap();
else
 do_something_expensive();
```
- **Generated Code**



# Overview

- Build expressions for jump conditions
  - trivial for high-level languages, but not for machine code
- Compare those expressions using a solver library (Omega)
- Use implications to generate new ILP constraints which exclude the infeasible paths



# Analysis

- Forwards analysis
- Analyses all conditional jumps where the contents of the condition register are still unknown after the value analysis
- Creates backward slices with the condition register as the initial target



# Slices

- Slice: set of program points which directly or indirectly influence the values at a given program point

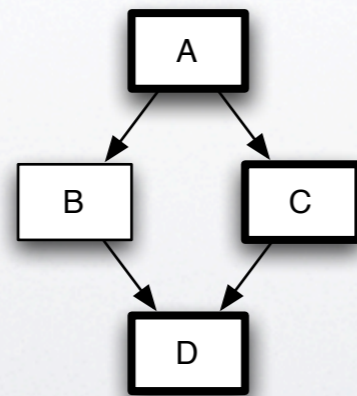
- Example:

```
add r5, r3, r4
cmpi cr0, 0, r5, 0
bc 0xd, cr0, 0x2c.t
```

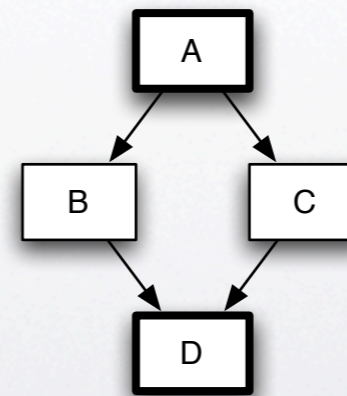


# Linear Slices

- A slice is called *linear*, iff the program points in the slice can be sorted, so that each program point is dominated by its predecessor



non-linear

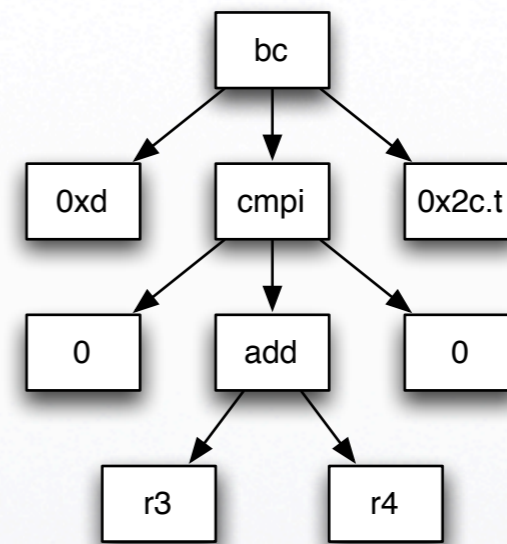


linear



# Slice trees

add r5, r3, r4  
cmpi cr0, 0, r5, 0  
bc 0xd, cr0, 0x2c.t



Constant Folding: (addi r5, 0, +16)  $\rightsquigarrow$  16



# Slice trees (2)

- **Nodes: instructions**
- **Leaves: registers, memory cells, immediates**



# Omega

- “Framework and algorithms for the analysis and transformation of scientific programs” by William Pugh and the Omega Project Team
- <http://www.cs.umd.edu/projects/omega/>
- Two Components
  - Omega-Test
  - Framework



# Omega-Test

- Presburger Arithmetic
  - First-order arithmetic over the natural numbers
  - $\forall, \exists, \neg, \wedge, \vee, +, -, =, \neq$
  - No multiplication!
  - Decidable



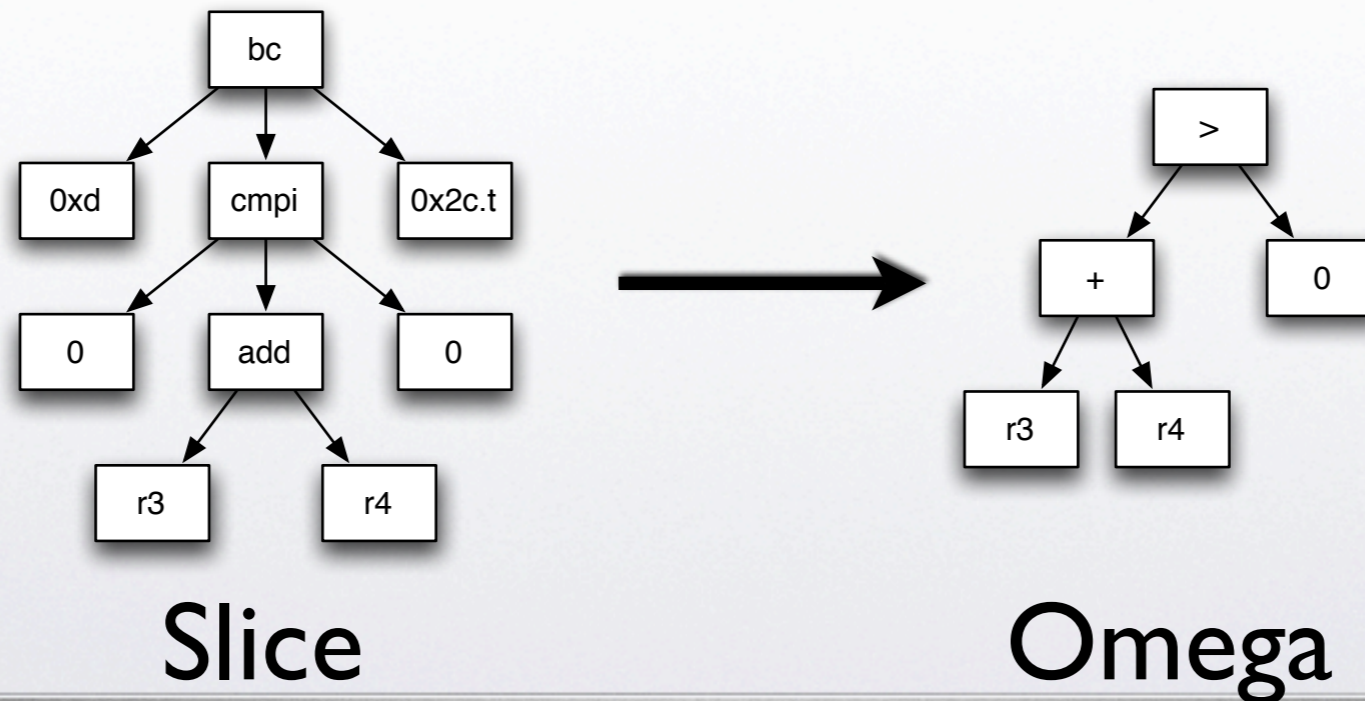
# Omega-Test (2)

- Simplifies and decides Presburger formulas
- Exponential runtime in the worst case
- Experimentally known to be fast enough for small problems



# Slices $\Rightarrow$ Omega

- Instructions in the slice tree are mapped to arithmetic and comparison operators according to their semantics





# Slices $\Rightarrow$ Omega (2)

- Instructions with unknown semantics are treated as symbolic functions
- `mul r3, r4, r5`  $\mapsto$  `mul(r3, r4, r5)`
- Operator Strength Reduction helpful for multiplication



# Slices $\Rightarrow$ Omega (3)

- Leaves
  - immediates become integer constants
  - Registers and memory cells become free variables



# Slices $\Rightarrow$ Omega (4)

- Given Omega trees for the basic blocks A and B
- $A \Rightarrow B$ ?
- $A \Rightarrow \neg B$ ?
- $\neg A \Rightarrow B$ ?



# Comparing Omega trees

- Compare Omega trees of slices with a common start point
- Stop slicing at the beginning of a function



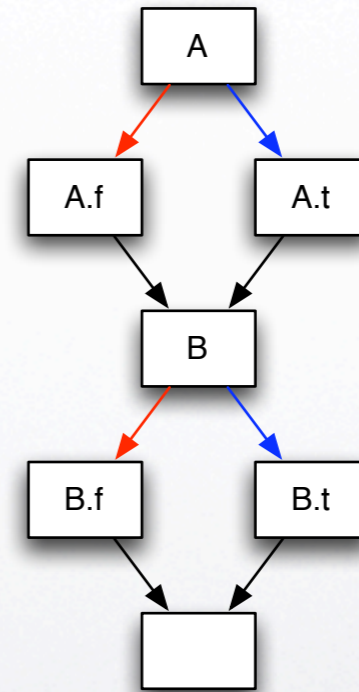
# Flow Constraints

- Relative execution counts of basic blocks
- Part of the ILP for the path analysis
- As AIS annotation:  
FLOW EACH ProgramPoint<sub>1</sub> / ProgramPoint<sub>2</sub>  
IS MIN min MAX max



# Flow-Constraints (2)

- $A \Rightarrow B: \quad A.t / B.t \cong I$
- $A \Rightarrow \neg B: \quad A.t / B.f \cong I$
- $\neg A \Rightarrow B: \quad A.f / B.t \cong I$
- $\neg A \Rightarrow \neg B: \quad A.f / B.f \cong I$
- Symmetric





# Implementation

- Implementation of the algorithm for the PowerPC architecture
- Additional architectures need their own mapping of the semantics
- Uses the generic slicer by Marc Schlickling
- Integrated into a special aiT version



# Evaluation

|                   | without flow constraints | with flow constraints |
|-------------------|--------------------------|-----------------------|
| Synthetic example | 3054 cycles              | 2290 cycles           |
| WBBC              | 2964 cycles              | 2961 cycles           |
| FCGU              | 1247 cycles              | 1221 cycles           |



# Future Work

- Other Architectures
- Non-linear slices
- Per-context flow constraints
- Dead code elimination
- Information about path exclusions can be used for other things besides flow constraints
  - May be beneficial to PAG-generated analyzers



# Summary

- A method to find path implications in the control flow graph
- Application in the path analysis
- Can improve the WCET prediction by eliminating infeasible paths
- May be used to improve PAG analyzers

